Software

IDC DOCUMENTATION

Message Subsystem

Contributors
John Coyne, Science Applications International Corporation
Deborah Dompierre, Pacific-Sierra Research Corporation
David Salzberg, Science Applications International Corporation
Michael Zatloukal, Science Applications International Corporation

Ordering Information
This document was issued by the Geophysical Systems Operation of Science Applications International Corporation (SAIC) as part of the International Data Centre (IDC) Documentation. The ordering number for this document is SAIC-98/3003, published May 1998. Copies of this document may be ordered by FAX: (619) 458-4993.

This document is cited within other IDC documents as [IDC7.4.2].

Message Subsystem

CONTENTS

Message Subsystem

FIGURES

Message Subsystem

TABLES

About this Document

This chapter describes the organization and content of the document and includes the following topics:

- Purpose
- Scope
- Audience
- Related Information
- Using This Document

About this Document

PURPOSE

This document describes the design and requirements of the Message Subsystem software of the International Data Centre (IDC). The software is a computer software component (CSC) of the Data Services Computer Software Configuration Item (CSCI). This document provides a basis for implementing, supporting, and testing the software.

This document is Issue 1 of an expected sequence of increasingly refined descriptions of the Message Subsystem software. This document supersedes the design and requirement descriptions contained in previous informal memoranda and Configuration Control Board proposals.

SCOPE

The Message Subsystem software is identified as follows:

Title:	Message Subsystem
Abbreviation:	(none)
Identification Number:	CSC 4.2
Version Number:	1

This document describes the architectural and detailed design of the software including its functionality, components, data structures, high-level interfaces, method of execution, and underlying hardware. Additionally, this document specifies the requirements of the software and its components. The information contained in this document is modeled on the Data Item Description for Software

Design Descriptions [DOD94a] and Software Requirements Specification [DOD94b].

AUDIENCE

This document is intended for engineering and management staff concerned with the design and requirements of all IDC software in general and of the Message Subsystem in particular. The detailed descriptions are intended for programmers who will be developing, testing, or maintaining the Message Subsystem.

RELATED INFORMATION

The following UNIX Manual Pages apply to the Message Subsystem software:

- *AutoDRM*
- *MessageAlert*
- *MessageFlow*
- *MessageFTP*
- *MessageGet*
- *MessageReceive*
- *MessageShip*
- *MessageStore*
- *ParseData*

The following documents complement this document:

- *IDC Database Schema* [IDC5.1.1]
- *Formats and Protocols for Messages* [IDC3.4.1]

See "References" on page 59 for a list of documents that supplement this document.

USING THIS DOCUMENT

The documentation of the IDC software is grouped within Category 7 of the overall documentation architecture, as charted on the Roadmap located on the pages preceding the Table of Contents. Within Category 7, the documentation is further subdivided into the same six CSCIs as the software architecture. The highlighted box represents this document, the Message Subsystem Software.

This document is organized as follows:

- Overview

  This chapter provides a high-level view of the Message Subsystem, including its functionality, components, background, status of development, and current operating environment.

- Architectural Design

  This chapter describes the architectural design of the Message Subsystem, including its conceptual design, design decisions, functions, and interface design.

- Detailed Design

  This chapter describes the detailed design of the Message Subsystem including its data flow, software units, and database design.

- Requirements

  This chapter describes the general, functional, and system requirements of the Message Subsystem. Traceability tables define how the general and functional requirements are met.

- References

  This section lists the sources cited in this document.

- Glossary

  This section defines the terms, abbreviations, and acronyms used in this document.

Conventions

This document uses a variety of conventions, which are described in the following tables. Table I shows the conventions (Gane-Sarson) for data flow diagrams. Table II shows the conventions for entity-relationship diagrams. Table III shows symbols used to indicate levels of the documentation hierarchy. Table IV illustrates typographical conventions. Table V explains certain technical terms that are not part of the standard Glossary, which is found at the end of this document.

TABLE I: DATA FLOW SYMBOLS

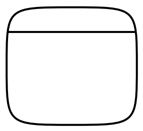
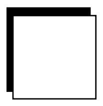
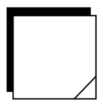
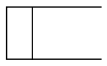
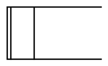
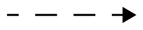
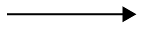
Description	Symbol
process	
external source or sink of data (left) duplicated external source or sink of data (right)	 
data store (left) duplicated data store (right)	 
control flow	
data flow	

Table II: Entity-relationship Symbols

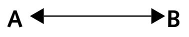
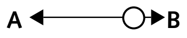
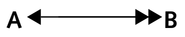
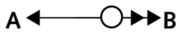
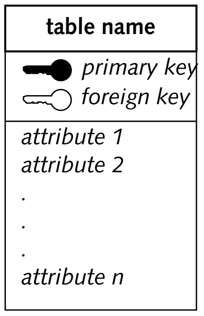
Description	Symbol
One **A** maps to one **B**.	
One **A** maps to zero or one **B.**	
One **A** maps to many **B**s.	
One **A** maps to zero or many **B**s.	
database table	 table name primary key foreign key attribute 1 attribute 2 . . . attribute n

Table III: Miscellaneous Symbols

Description	Symbol
category of documentation	●
subcategory of documentation	◆
document title	■

TABLE IV: TYPOGRAPHICAL CONVENTIONS

Element	Font	Example
database table	**bold**	**dataready**
database table and attribute, when written in the dot notation		**prodtrack**.*status*
database attributes	*italics*	*status*
processes, software units, and libraries		*ParseSubs*
user-defined arguments and variables used in parameter (par) files or program command lines		`delete-remarks` *object*
titles of documents		*Subscription Subsystem Software User Manual*
computer code and output	`courier`	`>(list ‘a ‘b ‘c)`
filenames, directories, and websites		`ars.scm`
text that should be typed in exactly as shown		`edit-filter-dialog`

Table V defines terms that are used in a specific context in this document. See the Glossary at the end of this document for a more general listing of terms, abbreviations, and acronyms.

TABLE V: TERMINOLOGY

Term	Description
fork	This UNIX system routine is used by a parent process to create a child process.
instance	An instance describes a running computer program. An individual program may have multiple instances on one or more host computers.
par (parameter) file	An ASCII file containing values for parameters of a program. Par files are used to replace command line arguments. The files are formatted as a list of `[token = value]` strings.
software unit or computer software component	These terms define an element of a Computer Software Configuration Item (CSCI) including a major subdivision of a CSCI or any of its contained subunits.

Overview

This chapter provides a general overview of the Message Subsystem software and includes the following topics:

- Introduction
- Functionality
- Identification
- Status of Development
- Background and History
- Operating Environment

Overview

INTRODUCTION

The software of the IDC acquires timeseries and radionuclide data from stations of the International Monitoring System (IMS) and other locations. These data are passed through a number of automatic and interactive analysis stages, which culminate in the estimation of location and in the origin time of events (earthquakes, volcanic eruptions, etc.) in the earth, including its oceans and atmosphere. The results of the analysis are distributed to States Parties and other users by various means. Approximately one million lines of developmental software are spread across six computer software configuration items (CSCIs) of the software architecture. Two additional CSCIs are devoted to non-developmental software and runtime data of the software. Figure 1 shows the logical organization of the IDC software. The Data Services CSCI receives, archives, and distributes data through the following computer software components (CSCs):

- Continuous Data Subsystem

  This software acquires timeseries data according to a standard protocol and forwards the data to external users [IDC3.4.2].

- Message Subsystem

  This software exchanges data in response to user requests. The data are formatted according to a standard protocol and exchanged through UNIX mail [IDC3.4.1]. This software also provides the interface to mail for the Retrieve and Subscription Subsystems.

- Retrieve Subsystem

  This software prepares messages, formatted according to the standard protocol, that retrieve segments of data from stations of the IMS auxiliary seismic network [IDC3.4.1]. The software also parses the response messages. The Message Subsystem exchanges the messages.

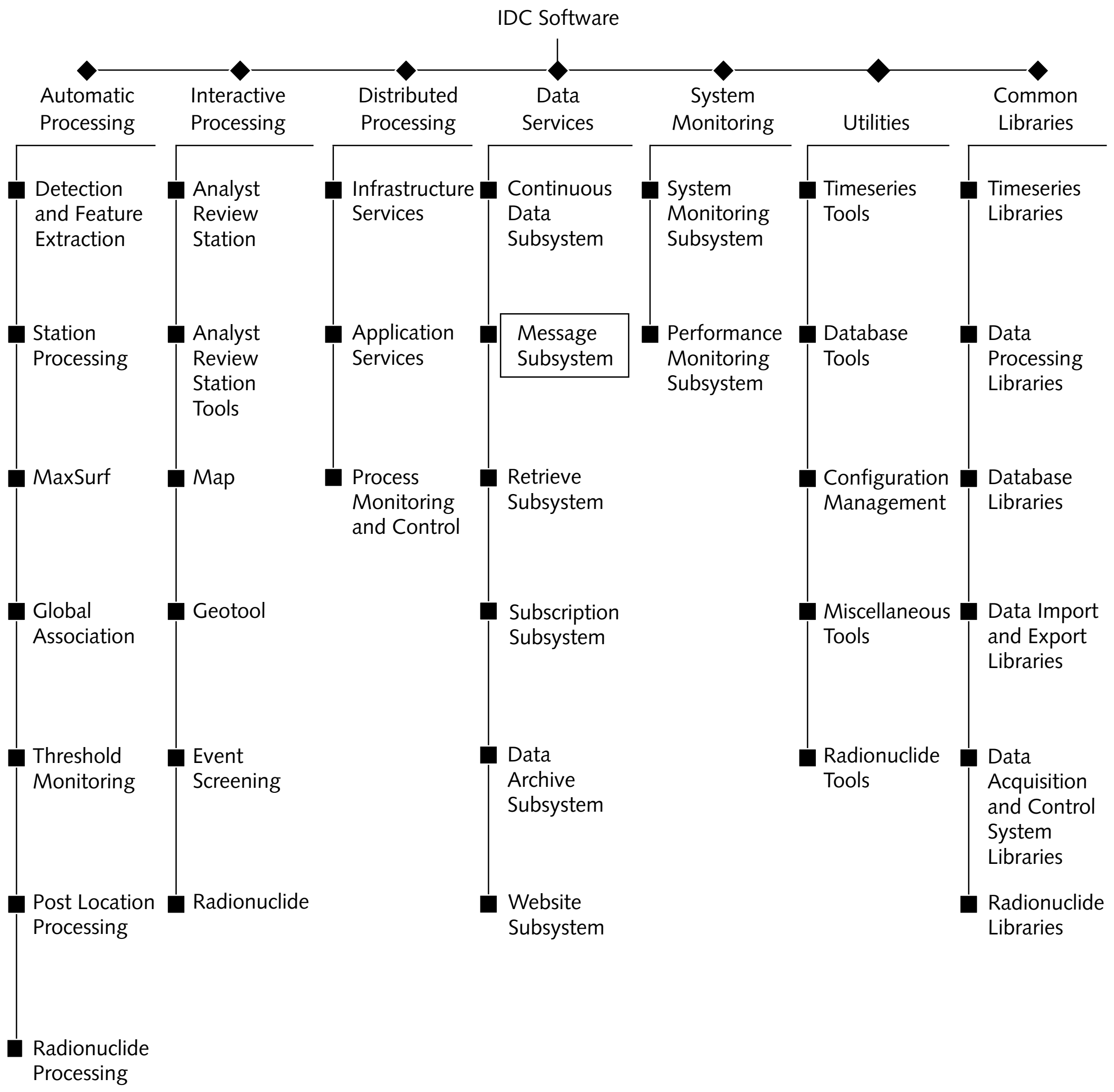


FIGURE 1. IDC SOFTWARE CONFIGURATION HIERARCHY

- Subscription Subsystem

  This software maintains a subscriber database and prepares the regular data products for delivery to subscribers. The Message Subsystem receives the subscription requests and delivers the subscription products.
- Data Archive Subsystem

  This software saves timeseries data to near-line and off-line archives and recovers data from the archives.
- Website Subsystem

  This software runs the IDC Web site.

The relationship of the Message Subsystem to the other components of the Data Services CSCI is indicated in Figure 2. This figure shows that the Message Subsystem (process 2) serves two roles. First, it accepts data from and provides data to a number of external users, represented by the boxes labeled b, c, d, h, and g. The Message Subsystem obtains data from and stores data within the Operations database. It also can provide data from the Archive database. The second role the Message Subsystem serves is to act as an agent for the Retrieve and Subscription Subsystems. In this role, the Message Subsystem is the interface with external users.

FUNCTIONALITY

The Message Subsystem enables authorized users to send a request for IDC products and have the products delivered. Requests are sent using email, and the products are delivered using email or the File Transfer Protocol (FTP). Requests are sent in the form of IMS Request Messages and the products are delivered in the same IMS format [IDC3.4.1]. The Message Subsystem is also used by the Request Subsystem to send requests to auxiliary seismic stations and to receive and parse the responses. The Message Subsystem parses the data messages from radionuclide stations and supplementary seismic bulletins. Incoming subscription messages are routed to the Subscription Subsystem, and subscription data messages are delivered by the Message Subsystem.

Users of the Message Subsystem must have access to a Simple Mail Transfer Protocol (SMTP) agent. An FTP application is needed to obtain voluminous products. It is also advisable to have software to parse the delivered data.

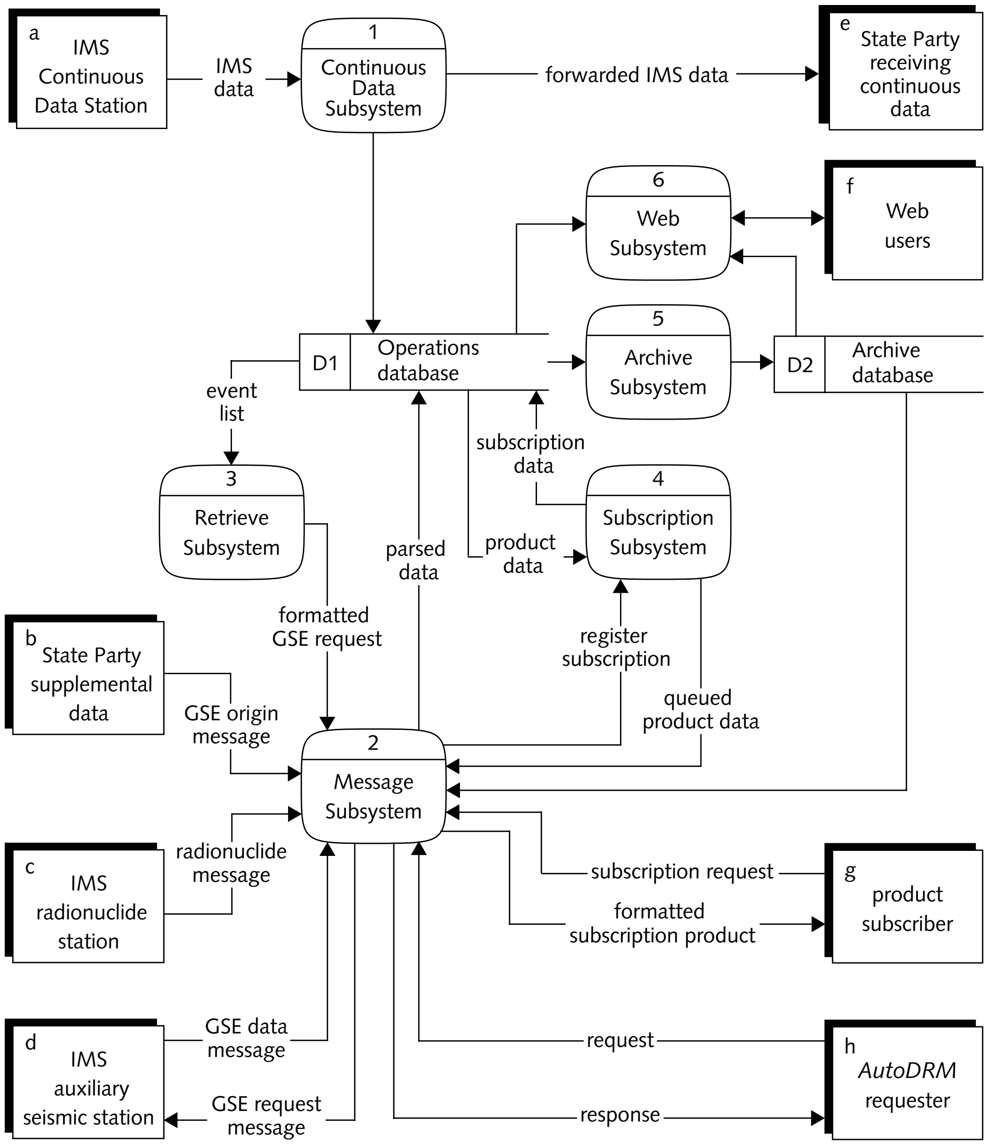


FIGURE 2. RELATIONSHIP OF MESSAGE SUBSYSTEM TO OTHER SOFTWARE UNITS OF DATA SERVICES CSCI

IDENTIFICATION

The Message Subsystem's components are identified in Table 1.

TABLE 1: COMPONENTS OF MESSAGE SUBSYSTEM

Component	System Version	Component Release Version
AutoDRM	6.0	Message Subsystem Release 6.0
MessageAlert	6.0	Message Subsystem Release 6.0
MessageFlow	not released	not implemented
MessageFTP	6.0	Message Subsystem Release 6.0
MessageGet	6.0	Message Subsystem Release 6.0
MessageReceive	6.0	Message Subsystem Release 6.0
MessageShip	6.0	Message Subsystem Release 6.0
MessageStore	6.0	Message Subsystem Release 6.0
ParseData	6.0	Message Subsystem Release 6.0

STATUS OF DEVELOPMENT

The Message Subsystem is almost completely developed. In the future, the Message Subsystem will verify users and will order requests to be processed in a more deterministic way. Additional products and formats will be supported. A graphically based control and monitoring program will be developed.

BACKGROUND AND HISTORY

David Corley of the Center for Monitoring Research (CMR) developed the Message Subsystem in 1995. The underlying software and database tables were modified extensively by Joe deBuzna and others in 1996.

The Message Subsystem, first used operationally in December 1995, currently is an element of the Prototype International Data Centre (PIDC) at the CMR in Arlington, Virginia, U.S.A. The software was also installed at the U.S. NDC (National Data Center) in Melbourne, Florida, U.S.A. in July 1997.

The International Data Centre of the Comprehensive Nuclear Test-Ban Treaty Organization (CTBTO IDC) in Vienna, Austria, is expected to install the Message Subsystem software.

The Operations staff of the IDC is responsible for operating the Message Subsystem. The level of operational involvement is expected to be slight, because the system is designed to run automatically. The software will be maintained by Science Applications International Corporation (SAIC) and the CMR through the Software Modification Request (SMR) process.

OPERATING ENVIRONMENT

The following paragraphs describe the hardware and commercial-off-the-shelf (COTS) software required to operate the Message Subsystem.

Hardware

The Message Subsystem software is designed to run on a UNIX workstation such as the SPARCstation 20/612. Typically, the hardware is configured with 64 MB of memory and a minimum of 2 GB of magnetic disk. The Message Subsystem must obtain other services (such as database access and mail) over its Ethernet connection to other computers. Figure 3 shows a representative hardware configuration.

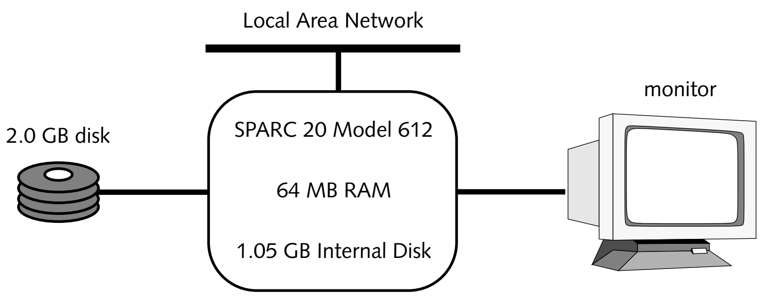


FIGURE 3. REPRESENTATIVE HARDWARE CONFIGURATION FOR MESSAGE SUBSYSTEM

Commercial-Off-the-Shelf Software

The Message Subsystem has been developed under Solaris 2.5 and ORACLE 7.2.3 and has been tested under Solaris 2.6. The software requires access to a SMTP mail transfer agent such as *sendmail*.

Architectural Design

This chapter describes the architectural design of the Message Subsystem and includes the following topics:

- Conceptual Design
- Design Decisions
- Functional Description
- Interface Design

Architectural Design

CONCEPTUAL DESIGN

The Message Subsystem provides the infrastructure for the exchange of intermittent data among the IDC, external users, and data providers. The data range from raw timeseries and radionuclide spectra to IDC products, such as reports and bulletins. The data are sent as messages, and adhere to the IMS 1.0 [IDC3.4.1] format (which descended from GSE 2.0 [GSE95a] and RMS 2.0 formats). The protocols used for data exchange are UNIX mail (*sendmail*) and FTP. The Message Subsystem supports both one time requests and a subscription service, which is described in [IDC7.4.4].

DESIGN DECISIONS

The following design decisions pertain to the Message Subsystem.

Programming Language

Each software unit of the Message Subsystem is written in the C programming language unless otherwise noted in this document.

Global Libraries

The software of the Message Subsystem is linked to the following (developmental) libraries: *libdrm*, *liblogout*, *libpar*, *libtime*, *libinterp*, *libaesir*, and *libgdi*.

The software of the Message Subsystem is also linked to the following COTS libraries: *libsql*, *libsqlnet*, *libncr*, *libclient*, *libcommon*, *libgeneric*, *libepc*, *libnlsrtl3*, *libc3v6*, *libcore3*, *libm*, *libF77*, and *libdl*.

Database

The Message Subsystem obtains data from the ORACLE database. Numerous tables in the database were created specifically for the Message Subsystem. These tables record information about messages and their state of processing.

Interprocess Communication (IPC)

The Message Subsystem does not utilize the message service of the Distributed Application Control System (DACS).

File System

The file system holds the run-time parameters of the Message Subsystem (par files). The Message Subsystem also reads and writes the formatted messages to the file system. Descriptors to these files are stored in the ORACLE database. Each programs' log file is written to the file system.

UNIX Mail

Messages are received and delivered by UNIX mail. The Message Subsystem stores a reference to each mail file and its status in the ORACLE database.

FTP

Products that are too large for reliable mail delivery are posted to an FTP directory; users are notified of the products' availability.

Web

A hypertext markup language (HTML) form allows users to submit data requests. For security reasons, the output of the form is sent to the Message Subsystem, and the response is delivered via email.

Design Model

The design of the Message Subsystem is primarily influenced by timeliness, flexibility, and reliability requirements. Although the system must process messages in a timely manner, a limited delay is acceptable to allow for IPC. Reliability and long-term tracking of the messages are also critical requirements. Thus, the system uses the IDC's database for both IPC and to archive the processing information.

Database Schema Overview

The Message Subsystem uses the ORACLE database for the following purposes:

- to record the state of message processing
- to describe flat files containing the actual incoming and outgoing messages
- to provide a link between data requests and responses

Table 2 shows the tables used by the Message Subsystem. The Name field identifies the database table. The Mode field is "R" if the Message Subsystem reads from the table and "W" if the system writes to the table. The Owner field indicates who has administrative control over the table. The "DBA" (Database Administrator) owns standard tables that are part of the core schema. The "MSG" (Message Subsystem) owns tables that are private to the Message Subsystem and are not accessed by any other software. The "RTR" (Retrieve Subsystem) owns the **request** table. The "RDBA" (Radionuclide Database Administrator) owns tables that are private to the radionuclide data processing software.

TABLE 2: DATABASE TABLES USED BY MESSAGE SUBSYSTEM

Name	Mode	Owner	Description
datauser	R/W	MSG	holds information about each user of the Message Subsystem
ftpfailed	R/W	MSG	holds tracking information on data messages transferred by FTP
ftplogin	R/W	MSG	holds FTP login and password data
msgaux	R/W	MSG	holds data for tracking unsuccessfully processed messages
msgdatatype	R/W	MSG	holds status data for individual data sections of data messages
msgdest	R/W	MSG	holds information about messages sent from the IDC
msgdisc	R/W	MSG	holds message information including date and time message was sent and received
poc	R	MSG	holds point of contact information for an organization
request	R	RTR	holds information needed to retrieve data from stations of the auxiliary seismic network
arrival	R	DBA	holds summary information on a seismic arrival
arrivalamp	R	DBA	holds amplitude measurements for arrival records
assoc	R	DBA	holds data associating arrivals with origins
beamaux	R	DBA	holds calibration information for beams
instrument	R	DBA	holds station calibration information
interval	R	DBA	holds information about the status of the Reviewed Event Bulletin (*ParseData*)

TABLE 2: DATABASE TABLES USED BY MESSAGE SUBSYSTEM (CONTINUED)

Name	Mode	Owner	Description
lastid	R/W	DBA	holds information on the last sequential value of one of the numeric keys
netmag	R/W	DBA	holds estimates of network magnitudes
origaux	R/W	DBA	holds additional data for supplementary events
gards_detectors	R	RDBA	contains detector overviews and characteristics
gards_notify	R	RDBA	contains contact information for specific occurrences
gards_stations	R	RDBA	contains station overview and characteristics
gards_userenv	R	RDBA	contains configurable variables used in radionuclide monitoring station software
gards_efficiency_pairs	R/W	RDBA	contains efficiency calibration pairs information associated with samples
gards_energy_pairs	R/W	RDBA	contains energy calibration pairs information associated with samples
gards_resolution_pairs	R/W	RDBA	contains resolution calibration pairs information associated with spectral pulse height data
gards_sample_data	R/W	RDBA	contains data from sample, blank, and calibration pulse height data messages
gards_alerts	W	RDBA	contains data describing all radionuclide alert messages received
gards_data_log	W	RDBA	contains data describing all radionuclide data messages received
gards_dose_data	W	RDBA	contains station average dose rate information during specific time intervals

TABLE 2: DATABASE TABLES USED BY MESSAGE SUBSYSTEM (CONTINUED)

Name	Mode	Owner	Description
gards_efficiency_cal	W	RDBA	contains efficiency calibration equation information associated with samples
gards_energy_cal_orig	W	RDBA	contains energy calibration equation information associated with samples
gards_environment	W	RDBA	contains atmospheric conditions and related sample information
gards_flow_data	W	RDBA	contains station flow rate data
gards_met_data	W	RDBA	contains station local meteorological data
gards_resolution_cal	W	RDBA	contains resolution calibration equation information associated with spectral pulse height data
gards_sample_aux	W	RDBA	contains auxiliary information related to raw sample data
gards_sample_cert	W	RDBA	contains overview information regarding certificate data corresponding to calibration pulse height data
gards_sample_cert_lines	W	RDBA	contains nuclide information regarding certificate data corresponding to calibration pulse height data
gards_sample_description	W	RDBA	contains description and comment text included within sample file
gards_sample_status	W	RDBA	contains spectral processing data
gards_total_effic	W	RDBA	contains detector total efficiency data calculated during calibration

FUNCTIONAL DESCRIPTION

The Message Subsystem consists of four functions: a function for importing messages, a function for routing messages, a function for processing messages, and a function for exporting messages.

Figure 4 shows the major functions of the Message Subsystem. Incoming messages are received by UNIX mail or FTP and are routed based on the message type. Messages of an unknown type are forwarded to an operator. Data messages are parsed, and the data (or pointers to the data, in the case of a waveform) are stored in the database. Subscription messages are routed to the Subscription Subsystem. Data request messages are fulfilled by *AutoDRM*. An `FTP_LOG` message initiates the *MessageFTP* process, which retrieves data from a remote site using the FTP protocol. Outgoing messages that fulfill a data request or subscription are exported using UNIX mail or FTP. The Retrieve Subsystem also uses the Message Subsystem to send data requests to stations of the auxiliary seismic network.

Importing Messages

Figure 4 illustrates the import function as follows: messages are imported using the email or FTP protocol. Each incoming message is written to the UNIX file system and an entry is added to the **msgdisc** table, which records the location, date, time, size, and status of the message.

Routing Messages

The routing function of the Message Subsystem passes each entry in the **msgdisc** table with a status of `RECEIVED` to the appropriate process based on the message type as recorded in the **msgdisc** table. For each type, a parameter specifies how the message shall be routed (which processing program) and what parameters to use.

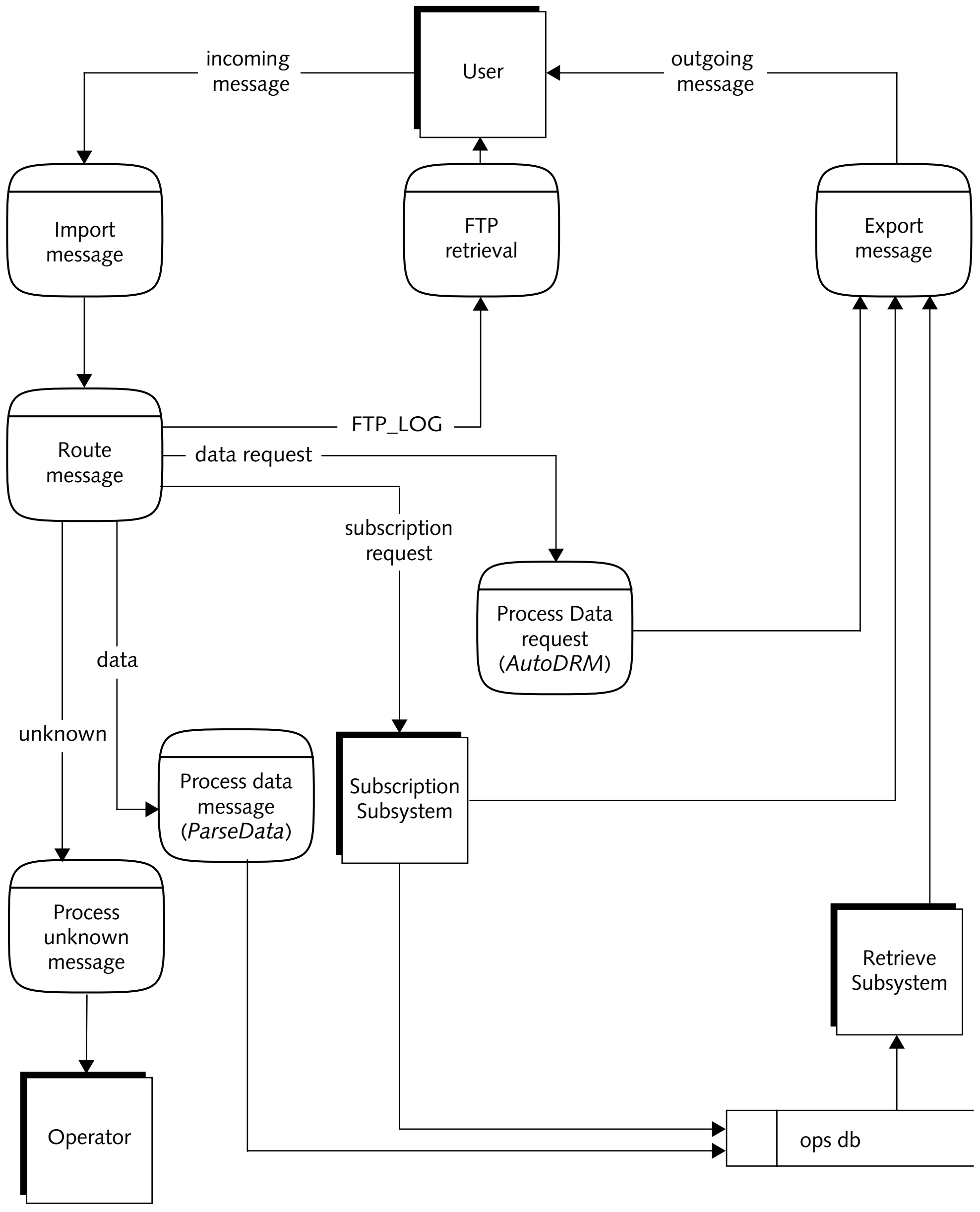


FIGURE 4. FUNCTIONAL DESIGN OF THE MESSAGE SUBSYSTEM SOFTWARE

Processing Messages

The processing function of the Message Subsystem reads and writes to a variety of tables. Incoming data messages are parsed by *ParseData*. Data from incoming timeseries data messages are written to the following tables: **msgdatatype**, **origin**, **origaux**, **origerr**, **outage**, **remark**, **netmag**, or **wfdisc**.

Data from incoming radionuclide data messages are written to the following tables: **gards_alerts**, **gards_dose_data**, **gards_flow_data**, **gards_met_data**, **gards_sample_data**, **gards_sample_aux**, **gards_sample_description**, **gards_sample_status**, **gards_environment**, **gards_energy_cal_orig**, **gards_energy_pairs**, **gards_resolution_cal**, **gards_resolution_pairs**, **gards_efficiency_cal**, **gards_efficiency_pairs**, **gards_total_effic**, **gards_sample_cert**, **gards_sample_cert_lines**, **gards_sample_cert**, and **gards_data_log**.

Incoming data requests are processed by *AutoDRM*, which queries many data base tables (**affiliation**, **arrival**, **arrivalamp**, **assoc**, **beamaux**, **ceppks**, **complexity**, **originamp**, **splp**, **spvar**, **evchar**, **flatdescription**, **flatproduct**, **instrument**, **netmag**, **origerr**, **origin**, **origaux**, **remark**, **sbsnr**, **sensor**, **site**, **sitechan**, **stamag**, **wfdisc**, **wfseg**, or **wftag**). `FTP_LOG` messages are processed by *MessageFTP*, which retrieves the specified file using the FTP protocol. FTP login information is read from the **ftplogin** table and FTP failures are recorded in the **ftpfailed** table.

Subscription messages are processed by the Subscription Subsystems. Tables used by Subscription Subsystem are described in the Subscription Subsystem [IDC7.4.4].

Table 3 maps data types to the database tables. The data types are defined in [IDC3.4.2], which explains the IMS 1.0 formats.

Exporting Messages

The export function of the Message Subsystem can be thought of as the mailing agent. Inputs are messages that need to be delivered, which have entries in the **msgdisc** and **msgdest** tables. The messages are sent using UNIX mail, or are placed in a local FTP directory, and a notification (`FTP_LOG`) message is sent. In both cases the *status* attribute in the **msgdest** table is updated to `DONE`.

TABLE 3: MAPPING DATA TYPES TO DATABASE TABLES

Database Table	Data Type
origin **origaux** **origerr** **(remark)** **netmag**	origin (supplementary seismic bulletin)
outage	outage
wfdisc **(outage)**	waveform
gards_alerts **gards_data_log**	alert
gards_dose_data **gards_data_log**	dose
gards_met_data **gards_data_log**	met
gards_flow_data **gards_data_log**	flow
gards_sample_cert_lines **gards_sample_data** **gards_efficiency_cal** **gards_efficiency_pairs** **gards_total_effic** **gards_energy_pairs** **gards_resolution_cal** **gards_resolution_pairs** **gards_sample_status** **gards_environment** **gards_energy_cal_orig** **gards_sample_aux** **gards_sample_description** **gards_sample_cert** **gards_data_log**	phd

INTERFACE DESIGN

This section describes how the Message Subsystem interfaces with other IDC systems, external users, and operators.

Interface with Other IDC Systems

The Message Subsystem sends subscription messages to the Subscription Subsystem. It also formats and delivers subscription data on behalf of the Subscription Subsystem. The Message Subsystem also provides a delivery service for requests originating from the Retrieve Subsystem and processes the incoming responses to those requests. All messages are exchanged using the email or FTP protocols.

Interface with External Users

The Message Subsystem was designed to interface with external users using messages in IMS 1.0 (GSE 2.0 or RMS 2.0) format. Messages are exchanged using email or FTP protocols. Waveform requests are generated by the Retrieve Subsystem, and are sent to stations of the auxiliary seismic network, where the requests are fulfilled and sent back to the Message Subsystem. When a product subscriber sends a subscription request, a subscription is initiated at the IDC. The Message Subsystem processes the products from that subscription and delivers them to the subscriber. When an *AutoDRM* user sends a request for data, the data are processed and a response message is returned. Both IMS supplemental data and radionuclide data are sent to the Message Subsystem when they become available at the data source.

Interface with Operators

The Message Subsystem exchanges data with system operators by forwarding copies of messages with unknown types, and by storing the state of the Message Subsystem in the database. The operator may monitor and maintain the Message Subsystem by using SQL commands directly or by using *MessageFlow,* the graphical user interface for the Message Subsystem. The Message Subsystem was designed to require minimum operator intervention.

Detailed Design

This chapter describes the detailed design of the Message Subsystem and includes the following topics:

- Data Flow Model
- Software Units
- Database Description

Detailed Design

DATA FLOW MODEL

The data flow model of the Message Subsystem is shown in Figure 5. All incoming email messages are received by *MessageStore* (process 1) and are placed in the temporary storage directory. *MessageReceive* (process 2) is a continually running process, which reads the files from the temporary storage directory, moves them to a more permanent disk area, and writes a pointer for the message in the **msgdisc** table in the database. *MessageReceive* also authenticates users based on data in the **datauser** table. *MessageGet* (process 4) finds new entries in the **msgdisc** table, orders them based on information in the **datauser** table, and routes each new incoming message to the appropriate destination. Incoming data messages are sent to *ParseData* (process 9), which parses the data into the database. Incoming data requests are routed to *AutoDRM* (process 8), which processes the request and writes the outgoing message. A reference for this outgoing message is written to the **msgdisc** table. *AutoDRM* processes requests generated by the Subscription Subsystem in a similar manner. Subscription messages are routed to the Subscription Subsystem. `FTP_LOG` messages are sent to *MessageFTP* (process 3), which retrieves the specified file using the FTP protocol and places the new message in the temporary storage directory. Each unknown message is forwarded to *MessageAlert* (process 7), which mails each failed message to the Message Subsystem operator.

In addition to outgoing data messages generated by *AutoDRM*, outgoing data request messages are generated by *MessageSend* (process 6). *MessageSend* is invoked by the Retrieve Subsystem. *MessageShip* (process 5) recognizes all outgoing messages and either mails the message or makes it available by FTP, depending upon the size of the message.

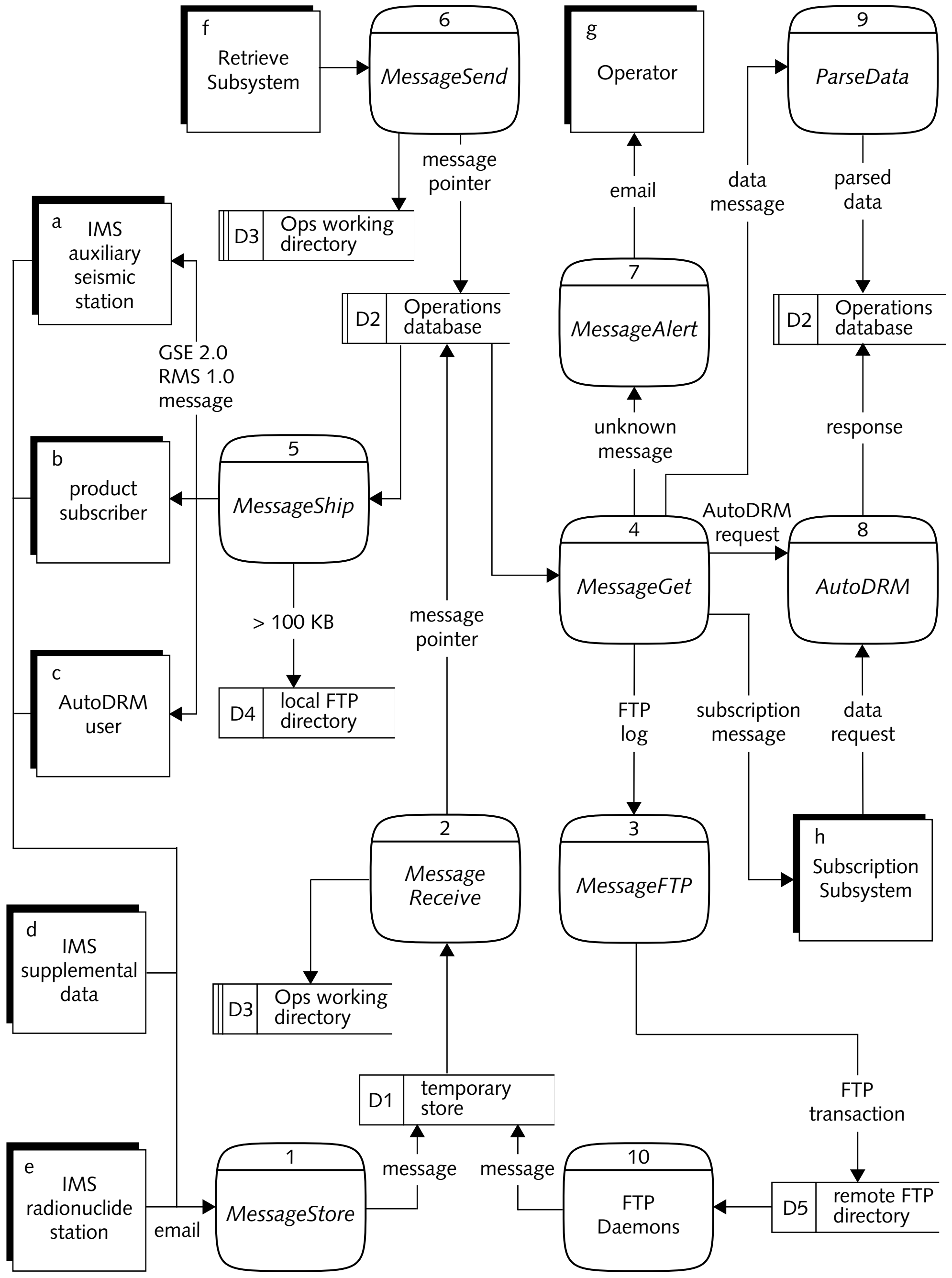


FIGURE 5. PROCESS VIEW OF MESSAGE SUBSYSTEM

SOFTWARE UNITS

The Message Subsystem consists of the following software units:

- *MessageStore*
- *MessageReceive*
- *MessageGet*
- *ParseData*
- *AutoDRM*
- *MessageFTP*
- *MesssageAlert*
- *MessageSend*
- *MessageShip*
- *MessageFlow*

The following paragraphs describe the design of these units, including any constraints or unusual features in the design. The logic of the software and any applicable procedural commands are also provided.

MessageStore

MessageStore accepts an incoming email message and stores it in an individual file in a temporary directory as efficiently as possible. The format of the message is not considered.

Input/Processing/Output

MessageStore reads a message from standard input and writes the message to the temporary storage directory. The path to the temporary storage directory may be given as the first argument on the command line, otherwise the directory `/tmp` is used. Any error messages are written to standard output.

Control

MessageStore is invoked when a mail message is received by the Message Subsystem mail alias. Mail is directed to *MessageStore* from *sendmail* by either a `.forward` file or other mail alias. *MessageStore* terminates after the message is written to the temporary storage directory. If the program runs successfully, the exit status is zero.

Interfaces

MessageStore interfaces with *sendmail* when it reads the message from standard input. The other interface is with the file system, where the message is written. *MessageStore* has no significant internal interfaces.

Error States

If the file system becomes full, the program will fail and the mail will be returned to the sender. If the temporary storage directory is not writable, the program will fail.

MessageReceive

MessageReceive reads incoming message files from a temporary directory, stores the messages in a permanent directory, and records the message pointer in the database.

Input/Processing/Output

MessageReceive retrieves incoming messages from the temporary storage directory and moves them to a permanent directory. *MessageReceive* authenticates users based on the digital signature in the message. If no digital signature is present, the address in the mail header will be compared with the entries in the **datauser** table. *MessageReceive* creates a **msgdisc** table entry with a *status* of `RECEIVED` and *msgtype* of either `DATA`, `SUBSCRIPTION`, `REQUEST`, `FTP_LOG`, or `UNKNOWN`. If the body of the incoming message contains `HELP` or `PLEASE HELP`, the file `help_file` is returned via email. If the parameter `help_file` is not defined, then *MessageReceive* forwards the message to *AutoDRM* to return its `help_file`.

All parameters are described in the *MessageReceive* manual page. All log information is written to the file system, as specified in the man page for *liblogout*).

Control

MessageReceive is one of three continuously running programs initiated at boot time with a shell script executed in the local directory of the Message Subsystem host computer.

After storing messages, *MessageReceive* sleeps for `sleep_time` seconds and then checks to see if any new messages have been written to the temporary storage directory. If so, the process is repeated, otherwise *MessageReceive* sleeps for another `sleep_time` seconds. *MessageReceive* will exit if it finds a file named .*end* in the temporary storage directory.

Interfaces

MessageReceive retrieves incoming messages that have been written in the temporary storage directory. A new **msgdisc** entry is made in the database and the message is copied to the `msgdir` directory. If the database cannot be opened, the message is copied to the `stagedir` directory. When the database is successfully opened, any messages under the `stagedir` directory are copied to the `msgdir` directory and appropriate **msgdisc** entries are recorded in the database. After the message is successfully copied to the `msgdir` or `stagedir` directory, the message is removed from the temporary storage directory.

Error States

If *MessageReceive* is unable to write to disk, it will exit and leave the file in the temporary storage directory for the next invocation of *MessageReceive*. If *Mes-*

sageReceive is unable to write to the database, it will write the file to an alternative directory and exit.

MessageGet

MessageGet forwards messages either to one of the processes *AutoDRM*, *Parsedata*, *MessageFTP*, *MessageAlert,* or to Subscription Subsystem.

Input/Processing/Output

MessageGet checks for messages with the status of `RECEIVE` in the **msgdisc** table. The output consists of changes in the **msgdisc** table and other processes that are forked by *MessageGet*. *Status* may be updated in the **msgdisc** table to `QUEUED` if the request is forwarded or to `status-parameter` on the first pass if the parameter `flush-queued` is enabled. For each message to be processed, *MessageGet* forks the appropriate child process. The exit status of the child process is captured. If the status is non-zero and the current **msgdisc** *status* will block the queue, the **msgdisc** *status* is updated to the value given by the par value `status_child_fails`.

All parameters are described in the *MessageGet* manual page. All log information is written to the file system, as specified in the man page for *liblogout*.

Control

MessageGet is one of three continuously running programs initiated at boot time with a shell script executed in the local directory of the Message Subsystem host. *MessageGet* terminates when it receives a fatal signal.

Interfaces

MessageGet finds records with status `QUEUED` and forks the appropriate process for each new message. Data requests are sent to *AutoDRM*; data messages are sent to *ParseData*; Subscription messages are sent to the Subscription Subsystem;

`FTP_LOG` messages are sent to *MessageFTP*; and unknown messages are sent to *MessageAlert.*

If the message type is recognized and if less than `max-queued` records of that type are already queued and if less than `max-parsing` of that type are already either running or queued, then the new record is forwarded to its destination and its status is changed to `QUEUED`. Otherwise, it is skipped until the next pass through the loop. Finally, *MessageGet* closes the database. If `loop` is enabled and `sleep-time` exceeds zero, *MessageGet* will repeat its procedures. Otherwise, it exits.

Error States

MessageGet will fail if it is unable to fork a child process. If *MessageGet* fails, it will kill the process and set the **msgdisc** *status* to a value indicating that the child failed. *MessageGet* relies heavily on the database. In the case of a failure to connect to the database, it will keep trying to connect until it is successful. If the database fails during operation of *MessageGet*, *MessageGet* will abort, and the pipeline operators will need to repair the database attributes, because *MessageGet* will be unable to accomplish that function.

ParseData

ParseData parses incoming data messages into the database.

Input/Processing/Output

ParseData interrogates the **msgdisc** table and then the message from disk. In the case of supplemental data, one or more records are written to the **origin**, **origaux**, **origerr**, **netmag**, and **remark** database tables, or, in the case of waveform data, a record is written to the **wfdisc** table, and the waveform file is written to disk.

In the case of radionuclide data, information is first written to the **msgdisc** table, and then the radionuclide file is written to disk. Then the *rms_pipeline* script is called. This script calls *rms_input*, which parses the message and inserts the data into the following database tables: **gards_alerts**, **gards_dose_data**, **gards_flow_data**, **gards_met_data**, **gards_sample_data**, **gards_sample_aux**, **gards_sample_description**, **gards_sample_status**, **gards_environment**, **gards_energy_cal_orig**, **gards_energy_pairs**, **gards_resolution_cal**, **gards_resolution_pairs**, **gards_efficiency_cal**, **gards_efficiency_pairs**, **gards_total_effic**, **gards_sample_cert_lines**, **gards_sample_cert**, and **gards_data_log**.

All parameters are described in the *ParseData* manual page.

All log information is written to the file system, as specified in the man page for *liblogout*.

Control

MessageGet forks a child process to initiate *ParseData,* which terminates after the data have been parsed into the database. On failure, *ParseData* returns a non-zero exit status.

Interfaces

ParseData reads the **msgdisc** table in the given database and finds the record with the given `msgid`. The message on disk pointed to by the **msgdisc** record is read and is parsed based on the type read from the message. As a result of this parsing the information from the message is placed in the database. *ParseData* will read through the entire file on disk and will parse all messages contained in the file. When parsing begins, the *status* attribute in the **msgdisc** table is changed to `PARSING`. If *ParseData* successfully parses the messages, *status* is changed to `DONE`. If any format errors are encountered in the message, *status* is changed to `PARSE-ERROR`. Any other error changes the *status* to `FAILED`.

Error States

ParseData rejects improperly formatted messages and mails them to the operator. If *ParseData* fails to connect to the database, *ParseData* returns a non-zero exit status to the parent process.

AutoDRM

AutoDRM provides automated email message responses to requests for data, which arrive by mail or are generated by the Subscription Subsystem.

Input/Processing/Output

AutoDRM provides automated email message responses to requests for data. The design of the *AutoDRM* message exchange is based on the protocols and formats adopted by the Group of Scientific Experts (GSE) for the GSE Technical Test 3 (GSETT3) requirements. The current standard, denoted IMS 1.0 is described in [IDC3.4.1].

A request message consists of a series of free-format command lines that provide information about the return message (response control lines), set the environment for subsequent request lines (environment lines), or specify the type of data that are to be returned within the limits of the environment (request lines). Message formats are described in [IDC3.4.1]. *AutoDRM* also formats the subscription products.

AutoDRM creates the data response and makes a **msgdisc** entry with *status* of `NEW` and a **msgdest** entry with a *status* of `PENDING`.

All parameters are described in the *AutoDRM* manual page. All log information is written to the file system, as specified in the man page for *liblogout*.

Control

For data requests that arrive by mail, *AutoDRM* is invoked as a child process by *MessageGet.* Upon updating the **msgdisc** and **msgdest** *status* fields with `NEW` and `PENDING`, the program exits with status equal to zero.

AutoDRM is also invoked as a child process of the *SubsProcess* software, a component of the Subscription Subsystem.

Interfaces

AutoDRM processes the request given in the specified message. In the course of fulfilling the request, many database tables are queried. The outgoing data message is written, along with the corresponding **msgdisc** and **msgdest** table entries.

Error States

The program will fail if it cannot access the mass storage device. If the file from the **wfdisc** cannot be read or opened, the *status* attribute of the **msgdisc** table is set to `STANDBY`. A reply message is returned to the requestor describing the problem. If the program cannot access the database, it exits with a non-zero status, and the parent process handles the error.

MessageFTP

MessageFTP is used to copy a file (using FTP) from remote sites to the IDC, upon receipt of an `FTP_LOG` message.

Input/Processing/Output

The input to *MessageFTP* is the `msgid` of an `FTP_LOG` message. *MessageFTP* attempts to retrieve the message specified in the `FTP_LOG` message. Successfully retrieved data are stored in the temporary storage directory, and the *status* attributes in **msgdisc** and **ftpfailed** (if the record exists) tables are updated to `DONE`. Otherwise, *status* of the **ftpfailed** and **msgdisc** tables is updated to `RETRY`. *MessageGet* periodically updates the *status* attribute of the **msgdisc** table for these types of messages from `RETRY` to `RECEIVED`, which results in another attempt by *MessageFTP*.

All parameters are described in the *MessageFTP* manual page. All log information is written to the file system, as specified in the man page for *liblogout*.

Control

MessageFTP is a child process spawned by *MessageGet*. When *MessageFTP* begins, it updates the *status* attribute of **msgdisc** to `RUNNING`. *MessageFTP* terminates after the FTP retrieval fails or succeeds.

Interfaces

MessageFTP retrieves the file specified in the `FTP_LOG` message passed by the `msgid` on the command line. An FTP session is initiated by the remote FTP site, and the specified file is written to the local temporary storage directory.

Error States

If *MessageFTP* cannot connect to or communicate with the database, it returns a non-zero exit status. If the remote FTP file does not exist or if the data are not retrieved after several attempts, *MessageFTP* updates the *status* in the **msgdisc** and **ftpfailed** tables to `FAILED`.

MessageAlert

MessageAlert sends email to specified users regarding an unrecognized message.

Input/Processing/Output

MessageAlert reads the **msgdisc** table in the database and finds the record with the given *msgid*. The **msgdisc** record points to the file containing the message on disk, which *MessageAlert* reads and mails to the users specified by `alert-email`.

When processing begins, *status* in the **msgdisc** table is updated to `RUNNING`. Once the mail is sent, the *status* is updated to `DONE`.

All parameters are described in the *MessageAlert* manual page. All log information is written to the file system, as specified in the man page for *liblogout*.

Control

MessageGet forks a child process to initiate *MessageAlert. MessageAlert* mails the operator the message and then exits with a zero exit status.

Interfaces

MessageAlert reads the specified message and mails it to the operator using the UNIX `mailx` command.

Error States

If *MessageAlert* cannot access the database or read the message from disk, it will exit with a nonzero status, and the parent process handles the error.

MessageSend

MessageSend supports the gathering of timeseries data from stations of the auxiliary seismic network. It accepts data specifications from the Retrieve Subsystem. It formats the specifications into messages that adhere to the standard protocol [IDC3.4.1]). It dispatches the message requesting data through UNIX mail.

MessageSend assumes that its message is received by an *AutoDRM* process.

Input/Processing/Output

MessageSend reads a message from standard input and writes the entire message to disk under the *msgdir* directory. Entries are made in the **msgdisc** and **msgdest** tables with the *status* of `NEW` and `PENDING` for the new message. Currently, *MessageSend* is called exclusively by *dispatch*, a software component of the Retrieve Subsystem. *Dispatch* writes the request portion of the message and then *MessageSend* creates the header and `STOP` lines of the message.

All parameters are described in the *MessageSend* manual page. All log information is written to the file system, as specified in the man page for *liblogout*.

Control

MessageSend is executed by *dispatch* and terminates after the message read from standard input is processed.

Interfaces

MessageSend is invoked as a child process of *dispatch*. It reads the body of the message from standard input and adds the header and `STOP` line to the message. The message is written to a permanent directory and records are written to the **msgdisc** and **msgdest** tables.

Error States

If *MessageSend* cannot access the database, it exits with a non-zero status and the parent process handles the error. This is also the result if the program cannot write the message to disk.

MessageShip

MessageShip mails messages to recipients.

Input/Processing/Output

MessageShip finds entries in the **msgdisc** and **msgdest** tables in the database where the *status* in the **msgdest** table is `PENDING` and the `msgids` in both tables are the same. If the method in **msgdest** is FTP, a notification message is mailed, and the message is copied to a standard directory of the file system on the machine specified by `ftp_host`. If the method in **msgdest** is mail, the message is mailed using the UNIX `mailx` command. In either case, the *status* in **msgdest** is updated to `DONE`.

All parameters are described in the *MessageShip* manual page. All log information is written to the file system, as specified in the man page for *liblogout*.

Control

MessageShip is one of three continuously running programs initiated at boot time with a shell script executed in the local directory of the Message Subsystem host.

If `loop` is set, *MessageShip* will run in loop mode. Otherwise, the program will query the tables, send any pending messages, and exit.

Interfaces

MessageShip processes messages whose entries in the **msgdisc** table have the *status* of `PENDING`. If the method in the **msgdest** table is FTP, the message is copied to the FTP directory and a notification message is mailed. Otherwise, the message is mailed with the UNIX `mailx` command.

Error States

MessageShip will fail if it is unable to write to the FTP directory. If the program cannot access the database, processing halts and the message is delivered when the database recovers.

MessageFlow

MessageFlow provides a graphical user interface for monitoring and controlling the Message Subsystem.

Input/Processing/Output

MessageFlow reads from database tables of the Message Subsystem and displays the current status of the Message Subsystem in a graphical user interface. Controls exist for stopping and starting the Message Subsystem, modifying the processing queues, and retrieving detailed information for a particular message.

Control

MessageFlow is started when the user invokes the program and terminates when the user selects the exit button within the graphical user interface.

Interfaces

MessageFlow reads from the Message Subsystem database tables and is capable of changing the *status* attributes in some tables. The temporary storage directory can be monitored. Signals can be sent to other components of the Message Subsystem, and *MessageFlow* can write an `.end` file to stop *MessageReceive*. *MessageFlow* can also restart the Message Subsystem.

Error States

MessageFlow will have reduced capability if it cannot connect to the database. If *MessageFlow* is not running on the same machine as the Message Subsystem, signals cannot be sent to the programs, and the Message Subsystem cannot be restarted.

DATABASE DESCRIPTION

This section describes the database design and schema. The Message Subsystem uses the database for all interprocess communication, primarily through *status* attributes. All code accesses the database using the Generic Database Interface (GDI) [And94].

Database Design

The Message Subsystem uses a database for storing, tracking, and queuing products. The entity-relationship model of the schema is indicated in Figure 6. The fundamental database table for the Message Subsystem is the **msgdisc** table. Each message has one entry in the **msgdisc** table; the entry is stored outside of the database as a flat file. **Msgid** is the primary key for the **msgdisc** table.

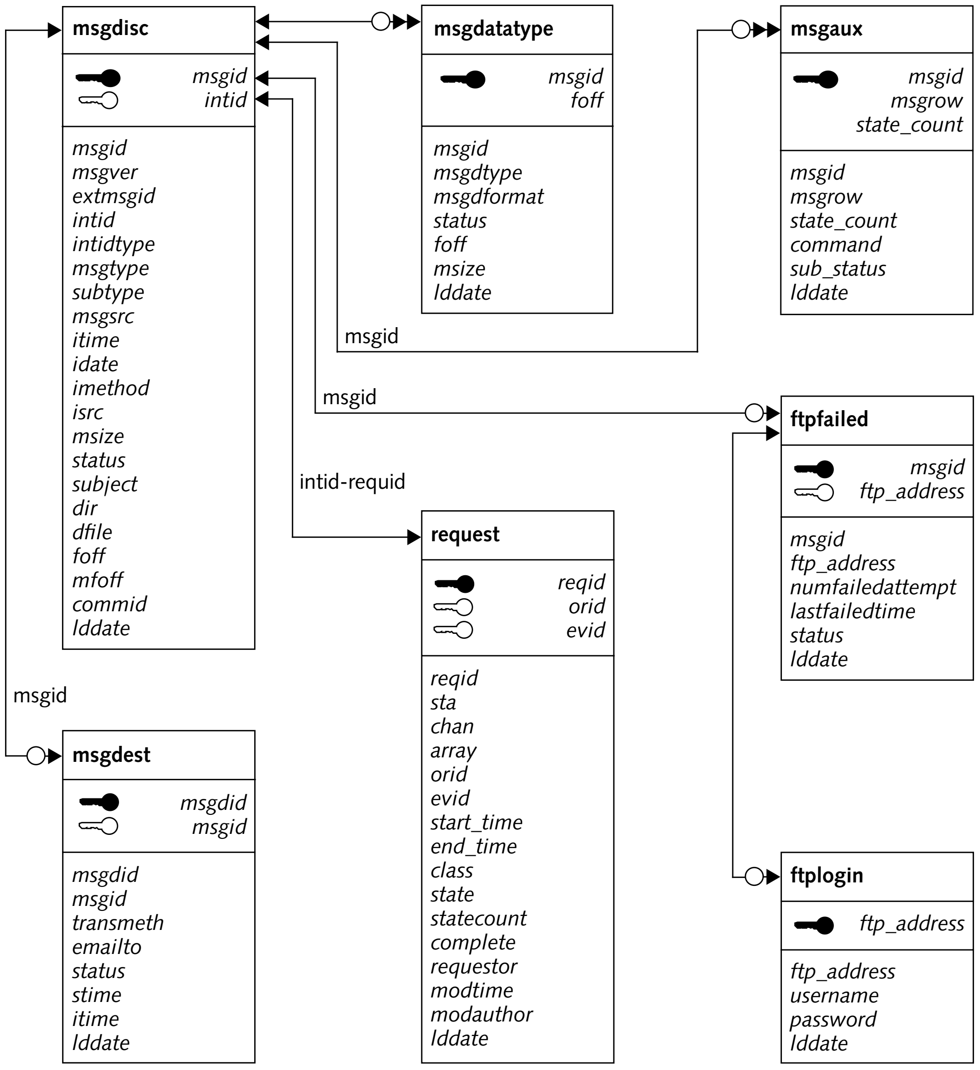


FIGURE 6. ENTITY-RELATIONSHIP OF MESSAGE SUBSYSTEM TABLES

The **msgdatatype** table is used to record the status of information within a message. For example, the status of each data section within a single data message is recorded in this table. When *MessageFTP* fails to retrieve a message using the FTP protocol, the failure is recorded in the **ftpfailed** table. FTP login information is recorded in the **ftplogin** table for those sites that do not support anonymous FTP. Each outgoing message also has an entry in the **msgdest** table; the entry specifies where the message should be sent and how it should be delivered. The **msgaux** table records failures in responding to requests for data from the IDC.

For each request generated by the Retrieve Subsystem, an entry is written to the **request** table. This request is linked to the **msgdisc** table by the *initid*, which is set equal to the *reqid* in the **request** table. The *intidtype* attribute is set to `REQID`.

The **datauser** table records information about each user of the Message Subsystem. The **poc** table records a single point of contact for each NDC or agency.

Database Schema

The Message Subsystem owns the following tables, which are detailed below:

- **datauser**
- **ftpfailed**
- **ftplogin**
- **msgaux**
- **msgdatatype**
- **msgdest**
- **msgdisc**
- **poc**

Datauser

The **datauser** table tracks authorized users of the IDC Message and Subscription Subsystems. Each user is identified by a (unique) *username* and *domain*, which must match all email headers. The *priority* attribute specifies the class of user, and *servicetime* is the last time a request from the user was processed. *Priority* and *servicetime* are considered when selecting the order in which requests will be processed. The *status* can either be active or inactive. Table 4 shows the **datauser** table. Primary and foreign keys are identified in Figure 6.

TABLE 4: DATAUSER

Column	Storage Type	Description
dataid	number(8)	identifier for the **dataready** table, set on the insertion
userid	number(8)	identifier for the **user** identifier
pocid	number(8)	point of contact identifier
username	varchar2(24)	user name from the incoming subscription message
domain	varchar2(48)	domain name from the incoming subscription message
status	varchar2(24)	status of this user
priority	number(2)	priority that this user has at the IDC
commid	number(8)	comment identifier
emaillimit	number(8)	maximum size of message (in bytes) that will be delivered via email
servicetime	float(53)	last time a request from that user was serviced
lddate	date	load date

Ftpfailed

The **ftpfailed** table contains the basic information required to track failures of transferring messages by FTP. *Numfailedattempt* is the number of times the FTP attempt failed, and the time of the last failure is given by *lastfailedtime*. Table 5 shows the **ftpfailed** table. Primary and foreign keys are identified in Figure 6.

TABLE 5: FTPFAILED

Column	Storage Type	Description
msgid	number(8)	identifier for the **ftpfailed** table
ftp_address	varchar2(64)	FTP address
numfailedattempt	number(4)	number of failed attempts

TABLE 5: FTPFAILED (CONTINUED)

Column	Storage Type	Description
lastfailedtime	float(53)	time of most recent attempt
status	varchar2(8)	status of FTP attempt (retry or failed)
lddate	date	load date

Ftplogin

The **ftplogin** table contains login information for sites that do not support anonymous FTP. Table 6 shows the **ftplogin** table. Primary and foreign keys are identified in Figure 6.

TABLE 6: FTPLOGIN

Column	Storage Type	Description
ftp_address	varchar2(64)	FTP address for auxiliary data
username	varchar2(16)	user name for FTP access
password	varchar2(16)	user password for FTP access
lddate	date	load date

Msgaux

The **msgaux** table contains the basic information required to track an unsuccessfully processed request message. The *msgid* and *msgrow* attributes identify the location in the message where the request failed. *State_count* records the number of times a request failed, which can be greater than two in the case of a hardware failure. Table 7 shows the **msgaux** table. Primary and foreign keys are identified in Figure 6.

TABLE 7: MSGAUX

Column	Storage Type	Description
msgid	number(8)	message identifier
msgrow	number(4)	line number in message
state_count	number(4)	number of failures
command	varchar2(24)	*AutoDRM* command that could not be processed
sub_status	varchar2(24)	cause of failure
lddate	date	load date

Msgdatatype

The **msgdatatype** table contains the basic information required to support data tracking of each data section in a message for both incoming and outgoing data messages. This table provides more details concerning each data section within a message, which is useful for compiling statistics. Table 8 shows the **msgdatatype** table. Primary and foreign keys are identified in Figure 6.

TABLE 8: MSGDATATYPE

Column	Storage Type	Description
msgid	number(8)	message identifier
msgdtype	varchar2(16)	data type of the data section within the message
msgdformat	varchar2(16)	general format of data that follows
status	varchar2(32)	status of the data section
foff	number(8)	file offset to beginning of data section
msize	number(8)	size of data section
lddate	date	load date

Msgdest

The **msgdest** table contains the basic information required to deliver an outgoing message. The *transmeth* attribute specifies if the message will be delivered by email or by FTP. The *emailto* specifies the address to which the data message or `FTP_LOG` message will be mailed. Table 9 shows the **msgdest** table. Primary and foreign keys are identified in Figure 6.

TABLE 9: MSGDEST

Column	Storage Type	Description
msgdid	number(8)	message delivery identifier
msgid	number(8)	message identifier of the response message created by *AutoDRM*
transmeth	varchar2(16)	method by which the response is to be delivered to the requester
emailto	varchar2(64)	email address to which the message has been sent
status	varchar2(32)	current status of the response message
stime	float(53)	time at which message was sent
itime	float(53)	time at which table entry was made
lddate	date	load date

Msgdisc

The **msgdisc** table contains the basic information required to specify files used for storing messages processed by the Message Subsystem. The *status* attribute records the status of each message.

Intid and *intidtype* can be used to link a **msgdisc** record with a record from another table. Currently, if *intidtype* is *reqid,* the *intid* will represent the request identifier of an entry in the **request** table. If *intidtype* is *msgid*, the *intid* will represent the *msgid* of an entry in the **msgdisc** table. Table 10 shows the **msgdisc** table. Primary and foreign keys are identified in Figure 6.

TABLE 10: MSGDISC

Column	Storage Type	Description
msgid	number(8)	the IDC unique identifier assigned to a message
msgver	varchar2(8)	identifies the Message Subsystem version number
extmsgid	varchar2(20)	the message identification string provided by the sender
intid	number(8)	either the locally generated *msgid* of an earlier **msg-disc** entry that evoked the creation of this **msgdisc** entry or the *reqid* from the **request** table of an internally generated request
intidtype	varchar2(16)	specifies the *intid* type
msgtype	varchar2(16)	specifies the message type
subtype	varchar2(2)	specifies the message subtype
msgsrc	varchar2(16)	the message source code
itime	float(53)	initial time message was received
idate	number(8)	initial date message was received
imethod	varchar2(8)	input method (email or FTP)
isrc	varchar2(64)	initial source of message
msize	number(8)	message size in bytes
status	varchar2(32)	status of message
subject	varchar2(64)	subject header from email message
dir	varchar2(64)	directory to find file
dfile	varchar2(32)	name of data file
foff	number(8)	byte offset of data segment within file
mfoff	number(8)	offset in bytes to beginning of message
commid	number(8)	comment identifier
lddate	date	load date

Poc

The **poc** (point of contact) table tracks a single point of contact for a particular site or agency. This table is only used when other attempts to contact an individual specified in the **datauser** table have failed. The *status* attribute can either be active or inactive (in the event that a point of contact is superseded). Table 11 shows the **poc** table. Primary and foreign keys are identified in Figure 6.

TABLE 11: POC

Column	Storage Type	Description
pocid	number(8)	message identifier
name	varchar2(20)	point of contact's name
email	varchar2(20)	point of contact's email address
address	varchar2(50)	point of contact's mailing address
telephone	number(8)	point of contact's telephone number
fax	number(8)	point of contact's fax number
affiliation	varchar2(16)	NDC or agency for point of contact
status	number(2)	point of contact's status
lddate	date	load date

Requirements

This chapter describes the requirements of the Message Subsystem and includes the following topics:

- General Requirements
- Functional Requirements
- System Requirements
- Requirements Traceability

Requirements

INTRODUCTION

The requirements of the Message Subsystem can be categorized as general, functional, or system requirements. General requirements are nonfunctional aspects of the Message Subsystem. These requirements express goals, design objectives, and similar constraints that are qualitative properties of the system. The degree to which these requirements are actually met can only be judged qualitatively. Functional requirements describe what the Message Subsystem is to do and how it is to do it. System requirements pertain to general constraints, such as compatibility with other IDC subsystems, use of recognized standards for formats and protocols, and incorporation of standard subprogram libraries.

GENERAL REQUIREMENTS

The Message Subsystem will meet the following general requirements:

1. The system will receive and parse unsolicited data contributed from authorized CTBTO sites. Unsolicited data is defined to be supplemental seismic data (NDC bulletins) and radionuclide data.
2. The system will receive and parse timeseries data solicited by the Retrieve Subsystem. The Retrieve Subsystem is defined to be the process that determines the data needs at the IDC and inserts requests for specific channels and intervals into a database table.
3. The system will receive and process request messages.
4. The system will receive and forward subscription messages.
5. The system will receive and process messages pertaining to the Continuous Data Subsystem.
6. The system will be extendable to facilitate parsing of new data types.

7. The system will be extendable to facilitate processing of requests for new data types.

FUNCTIONAL REQUIREMENTS

The requirements described in this section are categorized by function.

User Identification

The Message Subsystem is required to identify authorized users and to store user profiles and contact information as follows:

8. A user will be identified by a digital signature in the message following the MIME Object Security Services (MOSS) format (RFC-1848) [MOS95]. In cases where a message is not signed, the user will be identified by the email address in the message. Details of digital signature requirements are discussed in [Moo97].
9. The system will associate additional point-of-contact information with each user.
10. Each State Party may designate one person as a privileged user. The system will process requests from privileged users before processing other requests. The system will maintain a queue with a single slot for each privileged user. The earliest request by the user will be used to fill the slot in the queue. The queue of unprivileged users will be processed in a similar way.
11. The system will identify and exclude unauthorized users.

Message Tracking

The Message Subsystem is required to record the status of each message and link requests with data messages as follows:

12. All messages sent into the message system will be stored locally, and a record pointing to that file will be recorded in the database. Each record will contain a field, which specifies the message and its acceptance or the cause of the failure if the message was rejected.

13. Data from incoming data messages will be parsed and entered into the system after the message has been authenticated. If incoming data are not digitally signed, a lookup table will map addresses and data sources.
14. All messages will be archived for an indefinite period of time.
15. The system will be able to track any request message (incoming or outgoing) to the corresponding response data message.
16. The system will be able to track incoming data messages to the tables in which the parsed data are stored.
17. The system will generate statistics on the number, timeliness, and volume of messages by time, source, and content.
18. The system will reconcile the status of outstanding data requests by comparing outstanding requests with data that it has received.
19. Each message will be referenced by a unique identification number generated locally.

User Interface

The Message Subsystem is required to provide the following interfaces to users for submitting requests, receiving responses, and monitoring the system as follows:

20. The system will provide legacy support for GSE 2.0 and RMS 1.0 formats. (The `ASSOCIATED` command of GSE 2.0 is not supported, however.)
21. Users will be able to submit requests through email.
22. Users will be able to submit requests through a Web interface.
23. Users may specify a destination/delivery method with each request.
24. Every incoming request and non-data message will receive a response. No incoming data message will receive a response, unless an error is encountered or a return receipt is requested.
25. The user will be able to submit any type of message into the system and request a return receipt be mailed back to verify that the message was successfully received.

26. The user will have the ability to request the status of submitted requests. If the IDC reports that a response was emailed, the user may request that the message be resent.
27. The system will provide a mechanism that alerts authorized users of important system changes that will impact performance of the message handling system.

Operation and Logging

The Message Subsystem is required to operate and log its progress as follows:

28. The system will support multiple processing queues. At the highest level, request messages and data messages will have separate queues. Each queue may be further decomposed based on the age and data types contained in the message (see requirement 10).
29. System operators will be able to dynamically configure the processing queues described in requirement 28 without halting processing.
30. The system will log relevant information about processed messages including query data, recipient, time/date, and size.
31. The system will log all configuration/maintenance actions.
32. The system will rely on Internet-level security precautions to ensure the privacy of the log, all request messages, and corresponding responses.
33. Data made available by FTP will be stored in read-only directories on a per-user basis and will be removed after a fixed period of time after the user has been notified by email.
34. Logs will be kept for an operator-definable period of time.
35. The system will allow authorized individuals to monitor and control it in real time.
36. All components of the system will function properly even if multiple copies of that component are running at a given time.

Message Distribution

The Message Subsystem is required to deliver messages as follows:

37. Message data will be delivered through UNIX mail (for example, *sendmail*) to the email address associated with the request.
38. Message data may be delivered via FTP if the dataset to be returned is deemed too large to be sent by email, or if the user identifies FTP as the preferred delivery method. Local drop directories and FTP push will be supported as FTP delivery options.
39. The system will support the ability to automatically retrieve data using FTP upon receipt of an `FTP_LOG` message.
40. The system will recover and distribute products interrupted by a failure of hardware or software at the IDC.
41. The system will support the inclusion of customizable, product-dependent headers. At a minimum, headers will display the time and date when the product was prepared for delivery.
42. If a request returns no data, a `LOG` message indicating that no data exists will be sent.
43. The system will be capable of processing the following message types: request, subscription, data, and problem.
44. The system will distribute the following types of data messages and be capable of handling additional data types in the future: `STATION`, `COMM_STATUS`, the following time series products: `WAVEFORM`, `CHANNEL`, `INSTRUMENT`, `OUTAGE`, `RESPONSE`, `ARRIVAL`, `ORIGIN`, `EVENT`, `BULLETIN`, `COMMENT`, `STA_STATUS`, `CHAN_STATUS`, `AUTH_STATUS`, and the following Radionuclide products: `SAMPLEPHD`, `BLANKPHD`, `DETBKPHD`, `CALIBPHD`, `QCPHD`, `MET`, `DOSE`, `FLOW`, `ARMR`, `FPEB`, `RSR`, and `ALERT`.

SYSTEM REQUIREMENTS

The Message Subsystem will meet the following system requirements:

45. The system will use the IDC ORACLE database for all permanent data requiring transaction management.
46. The system will use command line arguments to pass run-time parameters to the application software. These arguments will be provided in par files, and standard IDC software will be used for reading and parsing these files.
47. The system will provide legacy support for GSE 2.0 and RMS 1.0 formats.

REQUIREMENTS TRACEABILITY

The following tables trace the requirements of the Message Subsystem to components and describe how the requirements are fulfilled.

TABLE 12: TRACEABILITY OF GENERAL REQUIREMENTS

	Requirement	How Fulfilled
1	The system will receive and parse unsolicited data contributed from authorized CTBTO sites. Unsolicited data is defined to be supplemental seismic data (NDC bulletins) and radionuclide data.	Data types for supplemental seismic data and radionuclide data can be parsed by *ParseData.*
2	The system will receive and parse timeseries data solicited by the Retrieve Subsystem. The Retrieve Subsystem is defined to be the process that determines the data needs at the IDC and inserts requests for specific channels and intervals into a database table.	The `WAVEFORM` data type can be parsed by *ParseData.*
3	The system will receive and process request messages.	Request messages are routed to and processed by *AutoDRM*.
4	The system will receive and forward subscription messages.	Subscription messages are routed to the Subscription Subsystem.

TABLE 12: TRACEABILITY OF GENERAL REQUIREMENTS (CONTINUED)

	Requirement	How Fulfilled
5	The system will receive and process messages pertaining to the Continuous Data Subsystem.	Requests for data to be retransmitted by the Continuous Data Subsystem are processed by *AutoDRM*.
6	The system will be extendable to facilitate parsing of new data types.	This requirement is not satisfied by the current release.
7	The system will be extendable to facilitate processing of requests for new data types.	This requirement is not satisfied by the current release.

TABLE 13: TRACEABILITY OF FUNCTIONAL REQUIREMENTS: USER IDENTIFICATION

	Requirement	How Fulfilled
8	A user will be identified by a digital signature in the message following the MIME Object Security Services (MOSS) format (RFC-1848) [MOS95]. In cases where a message is not signed, the user will be identified by the email address in the message. Details of digital signature requirements are discussed in [Moo97].	Message authentication will be performed by *MessageReceive.*
9	The system will associate additional point-of-contact information with each user.	Point-of-contact information is stored in the **poc** table.

TABLE 13: TRACEABILITY OF FUNCTIONAL REQUIREMENTS: USER IDENTIFICATION (CONTINUED)

	Requirement	How Fulfilled
10	Each State Party may designate one person as a privileged user. The system will process requests from privileged users before processing other requests. The system will maintain a queue with a single slot for each privileged user. The earliest request by the user will be used to fill the slot in the queue. The queue of unprivileged users will be processed in a similar way.	User privileges and the time of the previous response are stored in the **datauser** table and are considered by *MessageGet* when routing messages to the processing queues.
11	The system will identify and exclude unauthorized users.	*MessageReceive* will have the ability to identify and exclude unauthorized users.

TABLE 14: TRACEABILITY OF FUNCTIONAL REQUIREMENTS: MESSAGE TRACKING

	Requirement	How Fulfilled
12	All messages sent into the message system will be stored locally, and a record pointing to that file will be recorded in the database. Each record will contain a field, which specifies the message and its acceptance or the cause of the failure if the message was rejected.	A reference for each message is made in the **msgdisc** table. The *status* of each message can be derived from the *status* attribute in the **msgdisc** table and the *status* attribute of the corresponding **msgdatatype** records.
13	Data from incoming data messages will be parsed and entered into the system after the message has been authenticated. If incoming data are not digitally signed, a lookup table will map addresses and data sources.	This requirement is not satisfied by the current release.
14	All messages will be archived for an indefinite period of time.	Message archiving will be handled by the Archive Subsystem.

TABLE 14: TRACEABILITY OF FUNCTIONAL REQUIREMENTS: MESSAGE TRACKING (CONTINUED)

	Requirement	How Fulfilled
15	The system will be able to track any request message (incoming or outgoing) to the corresponding response data message.	The *msgid* of a request in the **msgdisc** table is recorded as the *initid* for the corresponding response in the **msgdisc** table.
16	The system will be able to track incoming data messages to the tables in which the parsed data are stored.	Incoming data messages are linked to the data tables by the **xtag** table, except for the waveforms, which are linked by the **wtag** table.
17	The system will generate statistics on the number, timeliness, and volume of messages by time, source, and content.	Values for compiling these statistics can be found in the **msgdisc** and **msgdatatype** tables.
18	The system will reconcile the status of outstanding data requests by comparing outstanding requests with data that it has received.	*WaveAlert* reconciles the status of outstanding data requests by comparing outstanding requests with data it has received.
19	Each message will be referenced by a unique identification number generated locally.	Each message is uniquely identified by the *msgid* in the **msgdisc** table.

TABLE 15: TRACEABILITY OF FUNCTIONAL REQUIREMENTS: USER INTERFACE

	Requirement	How Fulfilled
20	The system will provide legacy support for GSE 2.0 and RMS 1.0 formats. (The `ASSOCIATED` command of GSE 2.0 is not supported, however.)	Data messages in either format can be parsed by *ParseData*. Requests for data in either format will be supported by *AutoDRM*.
21	Users will be able to submit requests through email.	Requests may be sent to the Message Subsystem mail alias where they will be accepted by *MessageReceive.*
22	Users will be able to submit requests through a Web interface.	This requirement is not satisfied by the current release.

TABLE 15: TRACEABILITY OF FUNCTIONAL REQUIREMENTS: USER INTERFACE (CONTINUED)

	Requirement	How Fulfilled
23	Users may specify a destination/delivery method with each request.	Each request message may contain an email or FTP line that specifies the destination and delivery method.
24	Every incoming request and non-data message will receive a response. No incoming data message will receive a response, unless an error is encountered or a return receipt is requested.	This requirement is not satisfied by the current release.
25	The user will be able to submit any type of message into the system and request a return receipt be mailed back to verify that the message was successfully received.	This requirement is not satisfied by the current release.
26	The user will have the ability to request the status of submitted requests. If the IDC reports that a response was emailed, the user may request that the message be resent.	This requirement is not satisfied by the current release.
27	The system will provide a mechanism that alerts authorized users of important system changes that will impact performance of the message handling system.	The IDC will send a message of data type `COMMENT` to authorized users to inform them of any important system changes.

TABLE 16: TRACEABILITY OF FUNCTIONAL REQUIREMENTS: OPERATION AND LOGGING

	Requirement	How Fulfilled
28	The system will support multiple processing queues. At the highest level, request messages and data messages will have separate queues. Each queue may be further decomposed based on the age and data types contained in the message (see requirement 10).	Separate processing queues are supported by *MessageGet*.
29	System operators will be able to dynamically configure the processing queues described in requirement 28 without halting processing.	This requirement is not satisfied by the current release.
30	The system will log relevant information about processed messages including query data, recipient, time/date, and size.	Relevant information about messages is stored in the **msgdisc** and **msgdatatype** tables.
31	The system will log all configuration/maintenance actions.	This requirement is not satisfied by the current release.
32	The system will rely on Internet-level security precautions to ensure the privacy of the log, all request messages, and corresponding responses.	Access to logs and messages will be controlled by UNIX file permissions.
33	Data made available by FTP will be stored in read-only directories on a per-user basis and will be removed after a fixed period of time after the user has been notified by email.	The FTP directories will be configured so each user will only be able to access their own data.
34	Logs will be kept for an operator-definable period of time.	The operator may specify the duration for which logs will be retained.

TABLE 16: TRACEABILITY OF FUNCTIONAL REQUIREMENTS: OPERATION AND LOGGING (CONTINUED)

	Requirement	How Fulfilled
35	The system will allow authorized individuals to monitor and control it in real time.	*MessageFlow* will allow authorized individuals to monitor and control the Message Subsystem.
36	All components of the system will function properly even if multiple copies of that component are running at a given time.	This requirement is not satisfied by the current release.

TABLE 17: TRACEABILITY OF FUNCTIONAL REQUIREMENTS: MESSAGE DISTRIBUTION

	Requirement	How Fulfilled
37	Message data will be delivered through UNIX mail (for example, sendmail) to the email address associated with the request.	Data messages are delivered through UNIX mail by *MessageShip* to the address listed on the email or FTP line.
38	Message data may be delivered via FTP if the dataset to be returned is deemed too large to be sent by email, or if the user identifies FTP as the preferred delivery method. Local drop directories and FTP push will be supported as FTP delivery options.	*MessageShip* delivers data by FTP if the message is too large or if the user specifies the FTP option.
39	The system will support the ability to automatically retrieve data using FTP upon receipt of an `FTP_LOG` message.	*MessageFTP* can automatically retrieve data using FTP upon receipt of an `FTP_LOG` message.
40	The system will recover and distribute products interrupted by a failure of hardware or software at the IDC.	This requirement is not satisfied by the current release.
41	The system will support the inclusion of customizable, product-dependent headers. At a minimum, headers will display the time and date when the product was prepared for delivery.	This requirement is not satisfied by the current release.

TABLE 17: TRACEABILITY OF FUNCTIONAL REQUIREMENTS: MESSAGE DISTRIBUTION (CONTINUED)

	Requirement	How Fulfilled
42	If a request returns no data, a LOG message indicating that no data exists will be sent.	*AutoDRM* will respond with a LOG message if no data exists for a request.
43	The system will be capable of processing the following message types: request, subscription, data, and problem.	REQUEST messages are handled by *AutoDRM*. Subscription messages are handled by the Subscription Subsystem. DATA messages are handled by *ParseData*. PROBLEM messages are currently unsupported.
44	The system will distribute the following types of data messages and be capable of handling additional data types in the future: STATION, COMM_STATUS, the following time series products: WAVEFORM, CHANNEL, INSTRUMENT, OUTAGE, RESPONSE, ARRIVAL, ORIGIN, EVENT, BULLETIN, COMMENT, STA_STATUS, CHAN_STATUS, AUTH_STATUS, and the following Radionuclide products: SAMPLEPHD, BLANKPHD, DETBKPHD, CALIBPHD, QCPHD, MET, DOSE, FLOW, ARMR, FPEB, RSR, and ALERT.	*AutoDRM* currently responds to requests for the following data types: STATION, CHANNEL, WAVEFORM, INSTRUMENT, RESPONSE, ARRIVAL, ORIGIN, EVENT, and BULLETIN.

References

The following sources are referenced in this document or supplement it:

[And94] Anderson, J., Mortel, M., MacRitchie, B., and Turner, H., *Generic Database Interface (GDI) User Manual*, Science Applications International Corporation, SAIC-93/1001, 1994.

[DOD94a] Department of Defense, "Software Design Description," *Military Standard Software Development and Documentation*, MIL-STD-498, 1994.

[DOD94b] Department of Defense, "Software Requirements Specification," *Military Standard Software Development and Documentation*, MIL-STD-498, 1994.

[GSE95a] Group of Scientific Experts, *GSETT 3 Documentation, Volume One: Operations,* CRP/243, 1995.

[IDC3.4.1] Science Applications International Corporation, Pacific-Sierra Research Corporation, *Formats and Protocols for Messages*, SAIC-98/3004, PSR-98/TN1129, 1998.

[IDC3.4.2] Science Applications International Corporation, *Formats and Protocols for Continuous Data*, SAIC-98/3005, 1998.

[IDC5.1.1] Science Applications International Corporation, Pacific-Sierra Research Corporation, *Database Schema (Part 1 and Part 2)*, SAIC-98/3009, PSR-98/TN1127, 1998.

[IDC7.4.4] Science Applications International Corporation, *Subscription Subsystem*, SAIC-98/3001, 1998.

[MOS95] MIME Object Security Services (MOSS) format (RFC-1848), 1995.

[Moo97] Moore, *IMS Authentication Management and Requirements* document,1997.

[WGB97] Working Group B, *Operational Manual for the International Data Centre*, Preparatory Commission of the Comprehensive Test-Ban Treaty Organization, WGB/TL/44, 1997.

Glossary

A

Annexes

Formal additions to the Comprehensive Nuclear Test-Ban Treaty.

arrival

A signal that has been associated to an event. In the first instance, this is performed automatically by the Global Association (GA) software. Later in the processing, many arrivals are confirmed and improved by visual inspection.

B

bulletin

Chronological listing of event origins spanning an interval of time. Often, the specification of each origin, or event, is accompanied by th e event's arrivals, and sometimes with the event's waveforms.

C

child process

UNIX process created by the *fork* routine. The child process is a snapshot of the parent at the time it called *fork*.

CMR

Center for Monitoring Research.

Configuration Control Board

Organizational body that approves and releases new versions of software.

COTS

Commercial-Off-the-Shelf; terminology that designates products such as hardware or software that can be acquired from existing inventory and used without modification.

CSC

Computer Software Component.

CSCI

Computer Software Configuration Item.

CSP

Conference of States Parties; the principal body of the CTBTO consisting of one representative from each State Party accompanied by alternate representatives and advisers. The CSP is responsible for implementing, executing, and verifying compliance with the Treaty.

CTBT

Comprehensive Nuclear Test-Ban Treaty (the Treaty).

CTBTO

Comprehensive Nuclear Test-Ban Treaty Organization; Treaty User group that consists of the Conference of States Parties (CSP), the Executive Council, and the Technical Secretariat.

D

detection

Probable signal that has been automatically detected by the Detection and Feature Extraction (DFX) software.

E

email

Electronic mail.

exit status

Value returned at the completion of a UNIX command.

F

FTP

File Transfer Protocol; a method for transferring files between computers.

G

GB

Gigabyte.

GDI

Generic Database Interface.

GSE

Group of Scientific Experts.

GSETT-3

Group of Scientific Experts Third Technical Test.

H

HTML

Hypertext Markup Language; formatting language of the Web.

hydroacoustic

Pertaining to sound in the ocean.

I

IDC

International Data Centre.

IDC Operators

Technical staff that install, operate, and maintain the IDC systems and provide additional technical services to the individual States Parties.

IMS

International Monitoring System.

IMS Operators

Technical staff that operate and monitor the IMS facilities.

infrasonic

Pertaining to low-frequency (sub-audible) sound in the atmosphere.

Internet

World-wide network of computers linked by means of the IP protocol.

M

MB

Megabyte.

monitoring system

See IMS and RMS.

MOSS

MIME Object Security Services.

N

NDC

National Data Center.

O

online

Logged onto the network or having unspecified access to the Internet.

Operations Manuals

Treaty-specified, formal documents that describe how to provide data, receive IDC products, access the IDC database, and evaluate the performance of the IDC.

origin

Place and time of a seismic, hydroacoustic, or infrasonic event.

OSI

On-site Inspection.

P

parameter (par) file

ASCII file containing values for parameters of a program. Par files are used to replace command line arguments. The files is formatted as a list of `[token = value]` strings.

PIDC

Prototype International Data Centre.

PIDC System Developers

Contractors and other organizations who are developing and testing components of the PIDC technology.

Preparatory Commission

Preparatory Commission for the CTBTO; new international body funded by States Parties to prepare for implementation of the Treaty. This body will become the CTBTO after entry-into-force of the Treaty.

R

REB

Reviewed Event Bulletin; the bulletin formed of all events that have passed analyst inspection and quality assurance review. The REB runs 48 hours behind real time. The CTBTO is changing the name of this list to SEL3.

RMS

Radionuclide Monitoring System; the part of the IMS that monitors the atmosphere for radionuclides.

S

SEL1

Standard Event List 1; the bulletin created by total automatic analysis of continuous timeseries data. Typically, the list runs one hour behind real time.

SEL2

Standard Event List 2; the bulletin created by totally automatic analysis of both continuous data and segments of data specifically down-loaded from stations of the auxiliary seismic network. Typically, the list runs five hours behind real time.

SEL3

Standard Event List 3; the future name for the Reviewed Event Bulletin.

SMR

Software Modification Request.

SMTP

Simple Mail Transfer Protocol.

States Parties

Treaty user group who will operate their own or cooperative facilities, which may be NDCs.

T

taxonomy

Systematic arrangement; classification.

Treaty

Comprehensive Nuclear Test-Ban Treaty (CTBT).

Treaty Users

CTBTO and States Parties.

U

UN/CD

United Nations Conference on Disarmament.

W

Web

World Wide Web; a graphics-intensive environment running on top of the Internet.

workstation

High-end, powerful desktop computer preferred for graphics and usually networked.